Prouvé ? Et après ?

Faqing Yang, Jean-Pierre Jacquot

Prouvé ? Et après ?

Faqing Yang*, Jean-Pierre Jacquot*

*LORIA – Équipe DEDALE – Nancy Université
Campus scientifique, BP 239,
F-54506, Vandœuvre lès Nancy, France
{Prénom.Nom}@loria.fr

Résumé. Dans un contexte de certification, la relation entre méthodes formelles et logiciel « correct » n'est pas claire. Pour le formaliste, c'est la cohérence logique (preuve), pour le certificateur, c'est la bonne adéquation aux besoins et contraintes de l'usage prévu (validation). Nous présentons et discutons une analyse des difficultés et de leur résolution que nous avons rencontrées lors de la rédaction de spécifications en B événementiel. Nous proposons un processus de développement qui prend en compte la certification. Il est toujours fondé sur la notion de raffinement des propriétés fonctionnelles proposée par B événementiel. Les étapes de raffinement doivent être complétées par des sous-processus qui raffinent le modèle physico-mathématique pour l'amener à une forme acceptable par les outils de preuve, qui vérifient les contraintes non-fonctionnelles, principalement temporelles, et qui valident le comportement de la spécification.

1 Introduction

En 1993, dans un rapport commandité par la NASA, Rushby (1993) présentait une analyse des éléments pour, ou contre, l'emploi de méthodes formelles pour le développement de systèmes critiques. Les recommandations finales comportaient la nécessité d'encourager la croissance de l'utilisation des méthodes formelles dans l'industrie. Pourtant, malgré une amélioration des langages, des environnements et des outils supports, les méthodes formelles ressemblent à *l'Arlésienne* : on en parle beaucoup mais on ne les voit guère.

Nous sommes convaincus que les méthodes formelles peuvent et doivent être utilisées dans la plupart des développements logiciels. Nous croyons également que la résistance à l'usage de ces méthodes est moins liée à des questions théoriques ou techniques qu'à des questions méthodologiques. Comparons avec l'architecture.

La conception d'un bâtiment complexe est un mélange d'éléments développés avec des outils mathématiques sophistiqués et coûteux (la part « formelle »), d'éléments dessinés avec soin (la part « semi-informelle ») et d'éléments simplement esquissés (la part « informelle »). La structure du bâtiment est formalisée ; un modèle fin, coûteux, est établi pour vérifier des propriétés telles que la résistance sismique ou à la charge à la neige. Les réseaux et tuyauteries sont semi-formalisés : le modèle est nécessaire mais peut rester relativement grossier. Les cloisons internes relèvent de l'informel : on peut laisser aux maçons le soin de les placer et les réaliser sans risque. Toutes ces parts sont intégrées dans un *design* unique (le plan) qui peut être

lu selon différents points de vue. Ceci est possible car tous les intervenants comprennent ce que sont les différentes parts, comment elles interagissent, pourquoi on peut les traiter séparément, quelles sont les procédures à utiliser pour chacune, etc.

La certification (la recette) d'un bâtiment est également le fruit de plusieurs techniques articulées les unes avec les autres. La certification est aussi le résultat d'activités de validation qui se déroulent tout au long du projet, dès les premières esquisses. Les calculs de résistance par éléments finis sont effectués très tôt, des outils pour le calcul des tuyauteries sont employés pour valider la conception des réseaux, des maquettes (désormais virtuelles) sont construites pour validation par les commanditaires et les urbanistes. Les procédures d'assurance qualité sont elles aussi bien définies et bien appliquées. Au final, il y a un consensus sur ce qui fait qu'un plan est « correct » ou qu'une construction est « correcte ».

Nous sommes moins avancés pour les systèmes logiciels. Les méthodes, formelles ou non, sont souvent exclusives : elles ne se mélangent pas bien. De plus, les limites de chaque méthode sont imprécises. La validation des premières étapes d'un projet est encore un sujet délicat.

Ce papier est une discussion pragmatique sur les erreurs et les difficultés que nous avons rencontrées dans la rédaction de deux spécifications relatives au domaine des transports. Les deux ont été écrites en B événementiel. La première est une formalisation d'un algorithme de *platooning* ; la seconde est un modèle du domaine des transports. Les exemples discutés par la suite sont principalement extraits de la spécification du *platooning*. Plusieurs fois, nous avons découvert que la spécification était « incorrecte », de notre fait ou de celui d'autres rédacteurs. L'observation la plus intéressante est que cet état est déconnecté de la notion de spécification « prouvée ». Toutes les spécifications prouvées ne sont pas automatiquement correctes, et inversement. C'est une limite des méthodes formelles, B inclus.

Appréhender les limites est essentiel si nous voulons avoir confiance dans un morceau de logiciel développé formellement : nous devons nous assurer que la méthode a été utilisée à l'intérieur de son périmètre. Plusieurs techniques peuvent être imaginées à cet effet. Nous discutons quelques observations qui peuvent être intéressantes vis-à-vis de B événementiel.

La première observation a trait au problème des quantités numériques. Prouveurs, nombres réels et discrétisation ne font pas bon ménage. Il n'y a pas de notion d'approximation pour les preuves alors que les modèles physiques utilisent cette notion.

La deuxième observation concerne la qualité de la spécification en tant que modèle du problème. C'est la question de la validation des spécifications. L'animation de spécification apporte un élément de réponse.

La dernière observation traite du problème des propriétés temporelles. C'est un point crucial pour les systèmes critiques ; il faut le traiter très tôt dans la spécification.

La notion de raffinement utilisée en B classique ou événementiel est un argument important pour la promotion des méthodes formelles. Les outils tels que l'AtelierB ou Rodin implantent les traitements formels relatifs à la correction des raffinements, à la cohérence interne des spécifications et à la consistance fonctionnelle. Les activités de validation et de vérification des contraintes temporelles devraient elles-aussi être associées au raffinement. Nous devons viser à les intégrer dans un processus continu.

2 B événementiel et études de cas

2.1 B événementiel

B événementiel (Abrial (2009)) est une évolution de la méthode B (Abrial (1996)). Il est utilisé pour raisonner sur des systèmes complexes, distribués ou réactifs. La sémantique d'un modèle est donnée par les obligations de preuve qui sont générées pour montrer sa cohérence.

Spécification Abstraite. Une spécification abstraite en B événementiel est encapsulée dans une `MACHINE` qui possède un nom unique. Les variables du système sont déclarées dans une clause `VARIABLES`. Un `INVARIANT` définit l'espace des variables (leur type) et leurs propriétés fonctionnelles. Les événements de la partie `EVENTS` sont des substitutions. Leur sémantique correspond au calcul de la plus faible précondition (Dijkstra (1976)). Un événement est constitué d'une garde et d'un corps. Lorsque la garde est vraie, l'événement est *activé*. Le choix de l'événement à *déclencher* lorsque plusieurs sont activés est indéterministe. Le système a atteint un état terminal lorsqu'aucun événement n'est activé. On peut souhaiter (terminaison du calcul) ou non (*deadlock*) cette situation. De plus, il est possible d'introduire des `CONTEXT` où les données statiques du système (ensembles porteurs, constantes, fonctions, etc.) sont définies par leurs axiomes. Des obligations de preuve sont générées pour assurer la cohérence du modèle ; la préservation de l'invariant est l'une des plus importantes.

Raffinement. La progression vers une implantation s'effectue par un processus de raffinement. Le modèle abstrait est raffiné en un modèle plus concret. De nouvelles variables peuvent être introduites ; les variables abstraites peuvent être remplacées par des variables plus concrètes. Les substitutions dans les événements sont modifiées de façon conjointe. Une clause `WITH` permet d'exprimer le lien entre les paramètres d'un événement abstrait et ceux de l'événement concret. Il est également possible d'introduire de nouveaux événements. Ces nouveaux événements ne doivent pas empêcher indéfiniment les anciens d'être activés. La clause `VARIANT` exprime cette propriété. C'est une expression à valeur entière qui décroît strictement lorsqu'un événement concret désigné comme `convergent` est déclenché. Le raffinement est vérifié par la démonstration d'obligations de preuve spécifiques : préservation de l'invariant, décroissance du variant, non contradiction entre substitution concrète et abstraite, etc.

L'environnement Rodin[1] fournit les outils indispensables à la modélisation en B événementiel. Édition et preuves sont finement intégrées. Par ailleurs, Rodin possède des mécanismes d'extension et de configuration qui permettent de l'adapter à des domaines ou des méthodes de développement particuliers.

2.2 Spécification du *platooning*

Les recherches sur les systèmes de mobilité urbaine basés sur des petits véhicules électriques autonomes insistent sur l'importance d'un mode de fonctionnement spécifique : le *platooning*. Un *platoon* est un convoi de véhicules autonomes qui se déplacent de façon synchronisée. Dans un *platoon*, tous les participants suivent exactement les traces du premier

[1] http ://sourceforge.net/projects/rodin-b-sharp/

véhicule qui peut être éventuellement conduit par une personne. Le contrôle d'un *platoon* possède deux composantes : longitudinale, c'est-à-dire le maintien d'une distance inter-véhicules idéale, et latérale, c'est-à-dire l'asservissement à une trace virtuelle. Ces composantes peuvent être découplées et étudiées séparément (Daviet et Parent (1996)). La plupart des discussions qui suivent concernent la spécification de la composante longitudinale. La spécification considère essentiellement deux propriétés : (i) le modèle est cohérent à l'intérieur de ses limites, c'est-à-dire qu'aucune action ne le fait sortir de son domaine, et (ii) il garantit l'absence de collision entre véhicules.

Simonin et al. (2007) (resp. Lanoix (2008)) présentent une spécification d'un algorithme de contrôle longitudinal en B classique (resp. événementiel). L'invariant de base du modèle est que la distance entre deux véhicules est toujours supérieure à une valeur positive dite *critique*. Les auteurs affirment que la spécification est correcte, donc que l'algorithme est sûr.

La figure 1 montre les différents composants de la spécification. Le développement suit un processus de raffinement fondé sur le modèle Influence/Réaction (Ferber et Muller (1996); Ferber (1999)) : (i) tous les systèmes perçoivent, (ii) toutes les décisions sont calculées et (iii) tous les véhicules physiques se déplacent.

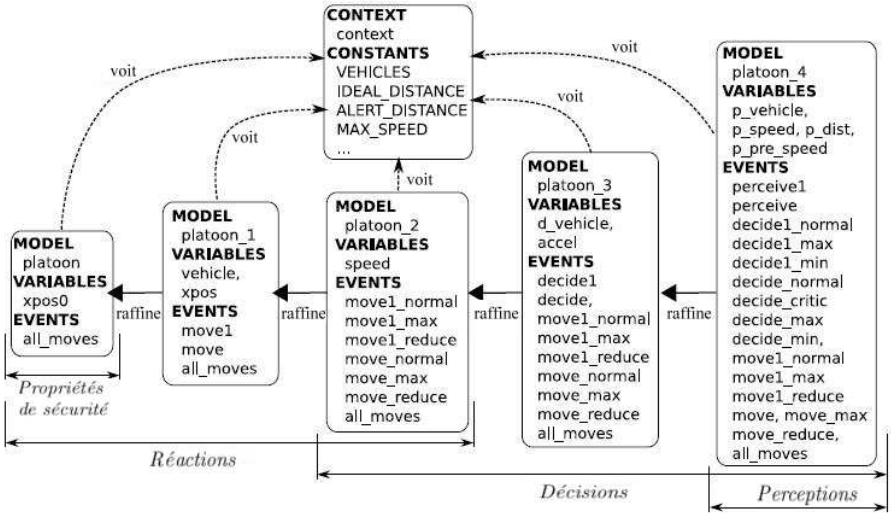

FIG. 1 – *Modèle en B événementiel du platooning(extrait de Lanoix (2008)).*

1. platoon : vue abstraite problème du *platooning*. Le mouvement des véhicules, qui correspond à l'étape de réaction, est décrit globalement. La propriété de sécurité, l'absence de collision, est exprimée à ce niveau.
2. platoon_1 : décomposition du mouvement global en un mouvement par véhicule.
3. platoon_2 : introduction de la vitesse des véhicules. Le mouvement d'un véhicule est alors défini par l'application d'une accélération passée au moteur.
4. platoon_3 : calcul de l'accélération de chaque véhicule. C'est la modélisation de l'étape de décision.

5. platoon_4 : perception par chaque contrôleur des informations nécessaires au calcul de l'accélération en fonction des règles de décision.

Un CONTEXT et ses raffinements contiennent les constantes, ensembles et fonctions cinématiques utiles.

3 Correct mais non prouvable

Le principe fondamental des méthodes formelles est l'association entre les notions de correction et de preuve. Faire confiance à une implantation qui ne peut pas être prouvée vis-à-vis de sa spécification ne serait pas raisonnable. Cependant, la puissance actuelle des prouveurs ne permet pas de rejeter systématiquement un logiciel qui ne respecte pas l'assertion précédente.

L'échec d'une preuve n'est pas toujours le signe d'une erreur dans le processus de développement. Le problème consiste donc à identifier les situations où une preuve « presque » complète peut être admise, au moins dans un premier développement.

Le *platooning* illustre cette situation. L'histoire est intéressante car elle met en évidence des propriétés fréquemment rencontrées dans les systèmes critiques.

3.1 Les buts admis

Les auteurs de la spécification ont affirmé que celle-ci était correcte puisque toutes les obligations de preuves avaient été vérifiées. Cependant, quatre buts étaient marqués comme *reviewed* (c'est-à-dire admis dans le jargon de Rodin). Il s'agit d'instances de deux propriétés mathématiques évidentes :

$$\neg\, c = 0 \quad \Longrightarrow \quad a * (b \div c) = (a * b) \div c \tag{1}$$

$$\neg\, d = 0 \quad \Longrightarrow \quad \neg\, 2 \times d = 0 \tag{2}$$

Le but 2 a été montré après que nous ayons trouvé une stratégie qui utilise une analyse par cas très artificielle. Le but 1 n'a pas été prouvé, et pour cause : il est incorrect ! Comme a, b et c sont des entiers, l'opérateur $\div$ dénote le quotient, pas la division.

Certes, une erreur a été commise, mais elle n'a pas de conséquence compte-tenu du contexte où les buts apparaissent. Ils interviennent dans des preuves qui impliquent des fonctions cinématiques complexes dans lesquelles les valeurs de a, b et c font que la différence entre les membres droit et gauche de l'équation est négligeable. Néanmoins, cette situation soulève un problème intéressant quant à la stratégie de modélisation.

3.2 Le problème de la discrétisation

Le cœur de la difficulté provient de la discrétisation des quantités cinématiques continues comme la position, la vitesse et l'accélération. Ne pourrait-on pas utiliser des valeurs « continues » ? Nous pensons que non.

Une première raison est d'ordre pratique. Les prouveurs actuels, dans le monde B, ne traitent que de nombres entiers. Même avec ces nombres « simples », les preuves sont souvent complexes et ramifiées. Les prouveurs efficaces traitant des nombres « réels » appartiennent à un futur lointain.

prouvé ? et après ?

Une seconde raison est plus profonde. Les systèmes logiciels sont par nature discrets. Parce que les sources de latence dans un véhicule automatisé sont nombreuses (délai d'acquisition des données, temps de réactions des actuateurs, temps de calculs par les processeurs, etc.), le système de contrôle opère à une fréquence relativement basse. Donc, le système réel fonctionnera comme si le temps était discret.

La méthode B vise à produire du code qui garantit la préservation des invariants fonctionnels. Nous devons donc introduire, à un moment ou à un autre, la discrétisation.

Il est préférable d'introduire cette caractéristique fondamentale au plus tôt dans la spécification : dès l'apparition des quantités continues dans le modèle. Naturellement, nous devons mettre en place des techniques et des stratégies pour traiter la discrétisation.

3.3 Quelques pistes de solutions

Les stratégies de modélisation avec des valeurs discrètes sont faciles à imaginer mais difficiles à utiliser dans la pratique :

1. réécrire les formules pour éviter les sous-buts avec égalité,
2. éviter l'opérateur $\div$.

Nous avons utilisé la première stratégie pour prouver les obligations qui introduisent les buts 1. La fonction suivante, déjà bien compliquée :

```
axm81 : new_xpos_max ∈ ℕ × 0..MAX_SPEED × MIN_ACCEL..MAX_ACCEL → ℕ
axm82 : ∀ xpos0, speed0, accel0 . (
    xpos0 ∈ ℕ ∧ speed0 ∈ 0..MAX_SPEED ∧ accel0 ∈ MIN_ACCEL..MAX_ACCEL
 ⇒ (
    (accel0 = 0 ⇒ new_xpos_max(xpos0↦speed0↦accel0) = xpos0 + MAX_SPEED)
  ∧
    (accel0 ≠ 0 ⇒ new_xpos_max(xpos0↦speed0↦accel0) =
        xpos0+MAX_SPEED−(((MAX_SPEED−speed0)∗(MAX_SPEED−speed0))/(2∗accel0)) )
  )
)
```

doit être réécrite sous une forme encore moins lisible :

```
axm81 : new_xpos_max ∈ ℕ × 0..MAX_SPEED × MIN_ACCEL..MAX_ACCEL → ℕ
axm82 : ∀ xpos0, speed0, accel0 . (
    xpos0 ∈ ℕ ∧ speed0 ∈ 0..MAX_SPEED ∧ accel0 ∈ MIN_ACCEL..MAX_ACCEL
 ⇒ (
    (accel0 = 0 ⇒ new_xpos_max(xpos0↦speed0↦accel0) = xpos0 + MAX_SPEED)
  ∧
    (accel0 ≠ 0 ⇒ new_xpos_max(xpos0↦speed0↦accel0) =
        xpos0 + ((MAX_SPEED−speed0)∗(MAX_SPEED−speed0))/(2∗accel0) +
        speed0∗((MAX_SPEED−speed0)/accel0) +
        MAX_SPEED∗(1−(MAX_SPEED−speed0)/accel0) )
  )
)
```

Lors de l'extension de l'algorithme de *platooning* au contrôle latéral, nous avons travaillé au niveau du modèle cinématique pour supprimer autant que possible les divisions dans les équations de description du mouvement. En conséquence, toutes les obligations ont pu être vérifiées avec les prouveurs disponibles dans Rodin.

Au delà de la division, les fonctions transcendantes ou trigonométriques posent également problème. Elles sont inconnues des prouveurs qui travaillent sur le domaine des entiers. Dans

le cas du modèle 2D du *platooning*, nous avons traité le problème au niveau du modèle mathématique. Dans les équations, nous avons remplacé ces fonctions par une approximation (développement de Taylor d'ordre limité).

Après ce conditionnement du modèle mathématique, l'écriture en B événementiel est assez simple. Toutes les obligations ont été prouvées. Cependant, le coût en terme de lisibilité des formules est très important.

4 Incorrect malgré une vraie preuve

Pendant l'élaboration de la spécification, l'équipe qui a fourni le modèle de contrôle et l'algorithme de *platooning* développait une simulation. Les deux développements ont évolué de façon conjointe ; les équipes se rencontraient souvent et une attention particulière a été portée au fait que la simulation implante le même modèle et les mêmes formules.

La simulation devait servir à calibrer certains éléments du modèle, comme la distance critique, et à observer des comportements non planifiés, comme l'oscillation de la vitesse des véhicules à l'intérieur d'un *platoon.* Cependant, le résultat le plus surprenant a été l'observation de quelques collisions, bien que la spécification soit totalement prouvée.

Plusieurs hypothèses ont alors été avancées pour expliquer les observations, mais aucune n'était convaincante. L'explication est apparue quand nous avons expérimenté avec la notion *d'animation*. Le processus d'animation et l'explication des collisions sont discutées ci-après.

4.1 Animer des spécifications

Le développeur d'une spécification formelle est confronté à deux questions : Est-ce que cette spécification est cohérente ? Est-ce qu'elle est un bon modèle de la fonction à implanter ? La puissance des cadres formels tels que B classique ou événementiel est d'apporter une réponse précise à la première question grâce à la notion d'obligation de preuve. Une spécification et ses raffinements sont cohérents quand toute les obligations ont été prouvées.

Répondre à la seconde question est beaucoup plus difficile. Ceci a été reconnu il y a longtemps, tout comme le potentiel de l'animation des spécification pour aider à répondre (Balzer et al. (1982)). Plusieurs travaux récents nous fournissent des outils pratiques qui permettent d'intégrer une activité d'animation dans le processus de développement (Bendisposto et al. (2008); Leuschel et al. (2001); Van et al. (2004); Schmid et al. (2000); Siddiqi et al. (1997)).

Animer une spécification en B événementiel est simple en principe. Il y a trois étapes :

1. l'utilisateur fournit des valeurs pour les constantes et les ensembles porteurs dans les contextes (explicitement ou en fixant le domaine exploré par les générateurs),
2. l'événement `INITIALIZATION` est déclenché pour mettre le système à l'état initial,
3. l'animateur entre dans une boucle :
   (a) calcul des gardes de tous les événements, activation de ceux dont la garde est vraie. L'argument des événements activés paramétrés est choisi arbitrairement,
   (b) le spécifieur choisit l'événement à déclencher ; les substitutions sont calculées,
   (c) vérifier l'invariant.

prouvé ? et après ?

La vérification de l'invariant est superflue dans le cas où la spécification a été totalement prouvée. Néanmoins, c'est une fonction très intéressante lorsqu'on utilise l'animation sur les spécifications non prouvées, par exemple pour s'assurer qu'une formule est un candidat plausible pour un invariant.

Dans la pratique, animer se révèle plus compliqué car les animateurs imposent des restrictions sur la forme des spécifications. Par exemple, les définitions non constructives, les ensembles infinis, les formules avec une quantification complexe ne peuvent pas être traitées. La spécification doit être transformée pour pouvoir être animée. Mashkoor et al. (2009) propose un processus de transformation qui est suffisamment peu coûteux pour être intégré à chaque étape de raffinement d'une spécification. L'idée est de sacrifier la prouvabilité tout en conservant les comportements.

4.2 Observations sur l'animation

La comportement à observer durant une animation est naturellement dépendant de la nature de la spécification. Un point particulièrement intéressant concerne l'arrêt, ou non, de l'animation. L'animation d'une spécification en B événementiel s'arrête lorsqu'aucun événement, à l'exception d'`INITIALIZATION`, n'est activé. Dans certain cas, c'est ce qu'on souhaite – le système a atteint un état terminal – dans d'autre cas, c'est une anomalie – le système est gelé.

Le *platooning* relève de la seconde catégorie de systèmes. Nous nous attendions à ce que la boucle de contrôle soit infinie, à tort. De fait, nous avons été capable de reproduire des situations dans lesquelles l'animation s'arrête. Notons que l'invariant est toujours valide lors de ces blocages. L'ironie de l'histoire veut que le premier blocage reproductible a été identifié lorsque le *platoon* stoppe (vitesse nulle) mais ne peut plus repartir.

Nous avons compris ce comportement inattendu lorsque nous nous sommes souvenus que les événements avaient été *conçus* pour maintenir l'invariant. Ainsi, un événement ne peut pas être activé si son déclenchement brise l'invariant. Donc, le blocage de l'animation signifie que tous les événements cassent l'invariant.

Les collisions sont alors expliquées. Il suffit de reformuler la situation : Quoi que le système fasse, l'invariant sera cassé ! Comme les véhicules sont encore en mouvement, ils vont entrer en collision. Manifestement, il manque quelque chose pour que la spécification soit sûre : les propriétés temporelles.

Rendre compte de ces propriétés dans le monde B est une affaire complexe. La section suivante discute ce point en détail.

5 Pas de solution simple pour les propriétés temporelles

B événementiel n'intègre ni la notion de temps, ni celle de logique temporelle. De fait, il n'y a même pas de notation pour dire que deux événements doivent se suivre.

Les propriétés temporelles et les contraintes d'ordonnancement sont parmi les points les plus importants pour les systèmes critiques. Aussi, les utilisateurs ont développé des usages standardisés pour simuler le temps et introduire des ordonnancements, voir Cansell et al. (2007) par exemple. Nous avons réutilisé certaines de ces techniques dans notre travail.

5.1 Un gros théorème

Assurer que le logiciel de contrôle du *platoon* est en mesure de réagir en n'importe quelle circonstance est une forme de propriété de vivacité. La technique standard pour garantir la vivacité d'une spécification B événementielle est d'introduire un théorème *ad-hoc* : la disjonction des gardes de tous les événements.

$$\text{vivacité} : \; G_{ev1} \vee G_{ev2} \vee \ldots \vee G_{evn}$$

où G_{evi} est la garde de l'événement i.

La construction de la formule est simple – elle peut être facilement automatisée – mais le résultat est très gros. Par exemple, *platoon_2* comporte 7 événements. En moyenne, la garde d'un événement est une conjonction de 5 formules. Ainsi, nous aboutissons à un théorème de 35 lignes. Nous laissons *platoon_4* avec ses 15 événements à titre d'exercice.

Cette taille énorme introduit plusieurs difficultés. Les spécifieurs ont l'occasion de commettre des erreurs de copie et la preuve est longue et pénible. Ces deux difficultés sont sérieuses mais peuvent être surmontées avec des procédures de relecture et de l'endurance. Une difficulté plus sérieuse surgit lorsque la preuve échoue.

À première vue, l'impossibilité de prouver une obligation est frustrante. Néanmoins, c'est un des meilleurs moyens de progresser vers une spécification correcte. En comprenant pourquoi une formule introduit un but non démontrable, nous acquérons une vision plus profonde de la spécification elle-même. Comme les contre-exemples produits par les *model-checkers*, les buts improuvables sont d'excellents guides vers l'expression correcte des bonnes propriétés.

Un des avantages majeurs de la méthode B est de casser la preuve de correction globale en plusieurs obligations de plus petite taille. Les échec des preuves sont ainsi plus faciles à analyser. Cette facilité disparaît quand les formules sont trop grosses.

Cette technique a permis de découvrir la source de l'erreur dans la spécification initiale : l'omission d'une contrainte liée au différentiel de vitesse entre véhicules dans la définition de la distance critique. Il faut noter que nous avons compris cette erreur après une réflexion complexe qui a impliqué l'intuition sur le modèle mathématique, l'observation et la reproduction des blocages avec l'animation ainsi que l'analyse de la formule. La taille a été un réel fardeau.

5.2 Agents indépendants et processus parallèles

La technique standard de vérification de la vivacité est utilisable sur le *platooning* grâce à une particularité de la spécification : elle modélise un algorithme séquentiel. La définition du problème précise que les véhicules sont implicitement synchronisés et exécutent perceptions, décisions et réactions en séquence. La séquence est explicitement codée dans les événements. D'autres schémas d'exécution sont envisageables.

Par exemple, il est possible d'imaginer un système dans lequel la perception soit découplée de l'action. Dans ce cas, les événements de perception pourraient être constamment activables. La technique utilisée ci-dessus serait alors inopérante.

Naturellement, un théorème de vivacité peut encore être défini : la disjonction des gardes des événements d'action. D'où les questions : Quels événements participent à ce théorème ? Comment s'assurer que tous les événements pertinents ont bien été inclus ?

La vivacité n'est pas la seule propriété qui intéresse l'activation des événements. Dans le modèle du domaine des transports, nous avons commis une erreur intéressante. Le modèle

prévoit plusieurs véhicules qui se déplacent indépendamment les uns des autres sur un réseau. Ils n'interagissent qu'aux intersections. Lors d'une étape, erronée, de développement, un seul véhicule à la fois pouvait se déplacer ; les autres devaient attendre qu'il ait atteint sa destination pour démarrer. Dans cette situation, il y a toujours un événement activé mais uniquement pour un véhicule. La situation correcte est qu'il doit exister un événement activé pour chaque véhicule en mouvement. Actuellement, nous ne savons pas comment noter cette propriété en B événementiel. Toutefois, l'animation permet de capturer immédiatement l'erreur.

5.3 Atteignabilité des états et des événements

L'autre notion classique associée aux systèmes critiques est l'atteignabilité. En B événementiel, elle correspond au fait qu'un certain événement sera activé. Pour rester dans les transports, on souhaite certainement s'assurer qu'un événement « atterrissage sûr » intervient pour tout avion qui a été paramètre d'un événement « décollage ».

B événementiel offre la notion de `VARIANT` pour exprimer des propriétés liées à cette problématique. L'utilité de cette notion est indiscutable, mais elle ne s'applique pas à toutes les situations. Clairement, elle est difficile à utiliser pour des systèmes basés sur une boucle infinie comme le *platooning*. Un autre problème se pose quand une notion d'attente explicite est introduite. Nous l'avons rencontré lorsque nous avons spécifié la notion de temps de trajet. Une interaction sûre aux intersections requiert qu'un véhicule attende que la voie soit libre. Un événement explicite `wait` est introduit à cet effet. Naturellement, le déclenchement de cet événement ne fait pas progresser le déplacement. Ceci détruit le variant raisonnable qui s'exprime comme la distance abstraite à la destination. La modélisation de protocoles de communication qui résolvent les collisions par l'activation de délais aléatoires mène à la même difficulté.

6 Un processus de développement étendu

6.1 Prendre en compte les propriétés non fonctionnelles

Comme nous l'avons vu, obtenir une « bonne » spécification n'est pas une tâche facile. En particulier, l'équation « spécification prouvée » égale « bonne spécification » ne tient pas. Le terme droit de l'équation doit être utilisé pour une spécification (a) qui est cohérente, (b) dont les propriétés fonctionnelles sont prouvées, (c) qui respecte les contraintes non fonctionnelles et (d) qui est un modèle raisonnable du problème. Dans le contexte des logiciels critiques ou certifiés, les méthodes formelles devraient fournir le cadre à l'intérieur duquel les « bonnes spécifications » sont élaborées.

B événementiel, en tant que complément au B classique, a été conçu autour d'un processus de développement : le raffinement. Les langages reflètent ce processus et les outils qui accompagnent la méthode B en font un candidat sérieux pour la construction de « vrais » logiciels. L'astuce est de couper la tâche gigantesque qu'est la preuve d'une implantation vis-à-vis de sa spécification en une longue suite de petites preuves. Les obligations génèrent beaucoup de formules à montrer mais chaque preuve reste relativement simple. Le coût global de la vérification reste maîtrisé. Donc, les points (a) et (b) reçoivent un traitement adéquat.

Une stratégie similaire doit pouvoir être employée pour les deux autres points. Le raffinement de B fournit la structure générale du développement ; les activités de preuve sont alors

complétées, à chaque étape, par des activités relevant de (c) et (d). Une étape de développement serait ainsi constituée de quatre « sous-processus » :

1. raffiner le modèle physico-mathématique. Ce sous-processus concerne l'expression de propriétés sous une forme que l'on puisse transmettre aux démonstrateurs. La discrétisation est étudiée à ce niveau.
2. définir et prouver le raffinement B. Ce sous-processus utilise les outils usuels associés à B événementiel.
3. animer la spécification. Ce sous-processus se décompose en trois activités. L'application des heuristiques pour rendre la spécification « animable » et l'exécution de l'animation sont purement techniques. La troisième activité concerne la création des scénarios qui doivent être mis en œuvre et observés.
4. analyser et raffiner les propriétés temporelles. Ce processus conduit à introduire des variants, des théorèmes de vivacité et à les prouver.

Les sous-processus doivent être réalisés dans l'ordre donné. Un modèle physique inadapté peut conduire à des formules certes correctes, mais non prouvables. Valider, même de façon partielle, un texte qui n'a pas été vérifié n'est pas pertinent : la spécification pourrait être incohérente. Enfin, l'animation peut suggérer les propriétés temporelles à contrôler. La figure 2 donne une vue graphique d'une étape de développement.

Morceler la validation et la vérification des propriétés non fonctionnelles de cette manière a plusieurs avantages. Le raffinement en B est bien compris, bien défini et bien outillé. Il permet de maîtriser la complexité d'un développement en limitant sa croissance à l'introduction d'un seul « nouveau détail » par étape. La preuve des obligations nous assure que ce nouveau détail est cohérent d'un point de vue fonctionnel. C'est donc un bon candidat pour organiser le développement global. Si l'analyse des autres dimensions, par l'animation et la preuve des théorèmes temporels, détecte une erreur, nous sommes au plus proche du point où elle est introduite. Si l'analyse se passe bien, nous pouvons passer en confiance au prochain raffinement.

De plus, nous pouvons espérer que la liste des raffinements successifs du modèle physico-mathématique, celle des scénarios et celle des contraintes temporelles seront des éléments probants qui simplifieront le travail de l'autorité de certification.

6.2 L'exemple, suite

Une fois découverte l'anomalie dans la spécification initiale, une part importante du travail était faite mais il nous restait à comprendre quelle erreur provoquait les blocages, quand elle avait été introduite et comment la corriger.

Nous avons rejoué le développement en suivant le processus décrit précédemment. Les résultats sur le développement initial sont résumés dans la table suivante :

	obligations	**comportement (animation)**	**propriétés temporelles**
platoon	prouvées	cohérent	vérifiées
platoon_1	prouvées	cohérent	vérifiées
platoon_2	assomptions incorrectes	incohérent	non vérifiées
platoon_3	prouvées	à faire	à faire
platoon_4	prouvées	à faire	à faire

prouvé ? et après ?

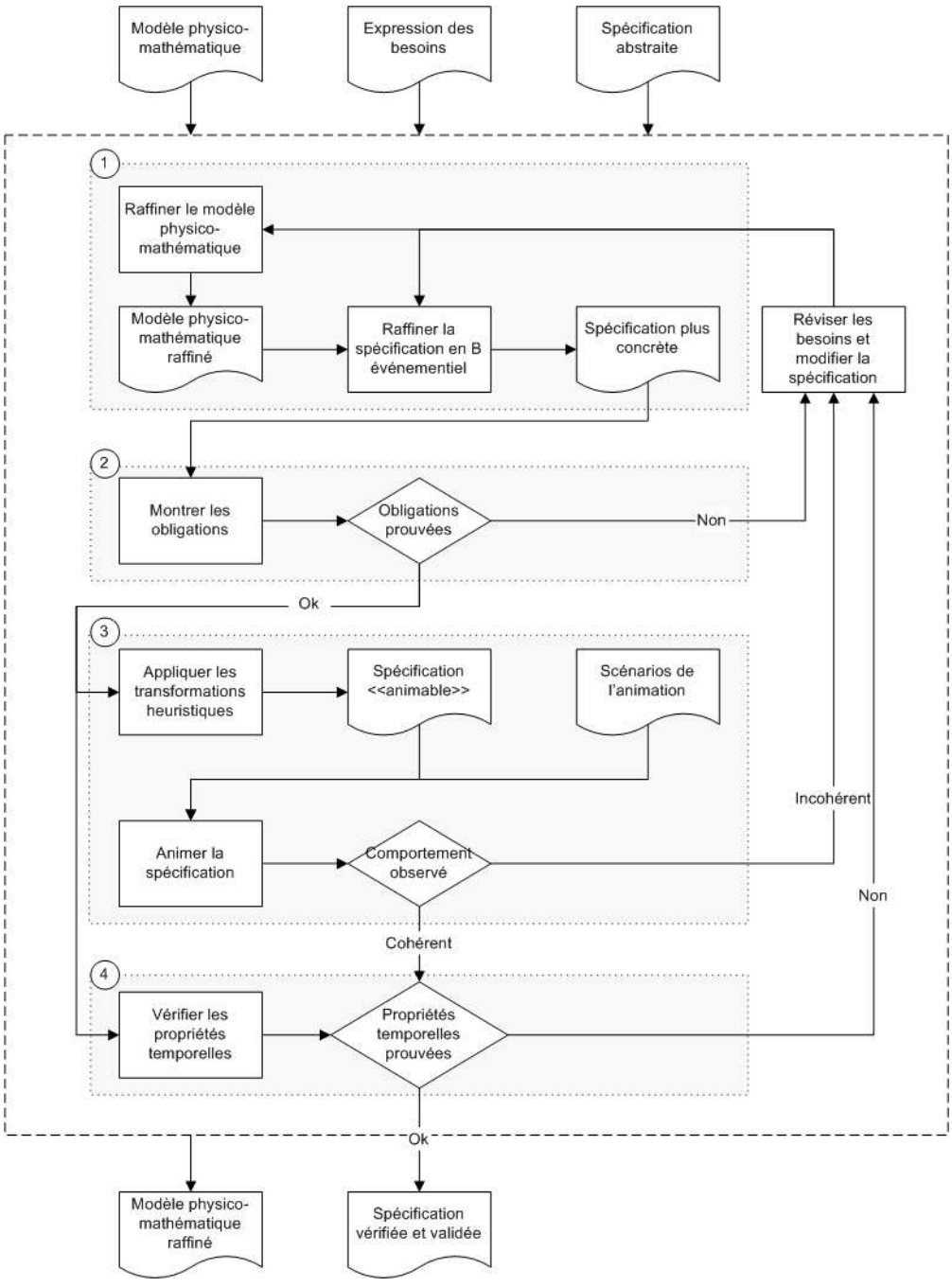

FIG. 2 – *Une étape du processus de développement.*

L'assomption incorrecte sur les sous-buts est la première erreur que nous avons découverte. À l'époque, nous pensions que l'anomalie dans le comportement pouvait provenir de là. Après que le modèle mathématique ait été retravaillé pour passer sous les fourches caudines de B, toutes les obligations ont pu être prouvées.

Malgré cette correction, le comportement était toujours anormal. Nous avons alors repris les animations en parallèle avec la preuve du théorème de vivacité. Bien que nous sachions que la preuve était impossible, nous espérions obtenir une intuition à partir des sous-buts non prouvables générés par le prouveur.

Notre enquête a finalement désigné comme coupable la modélisation de la non collision. Dans la spécification initiale (Lanoix (2008)), cette propriété est exprimée par :

INVARIANT
$\forall v.(\ v \in 2..\mathsf{VEHICLES} \Rightarrow \mathsf{xpos}(v-1) - \mathsf{xpos}(v) > \mathsf{CRITICAL\_DISTANCE}\ )$

où
- $\mathsf{VEHICLE} \in \mathbb{N}$: nombre des véhicules dans le *platoon*,
- $\mathsf{CRITICAL\_DISTANCE} \in \mathbb{N}$: la distance de sécurité critique entre véhicules,
- $\mathsf{xpos} \in 1..\mathsf{VEHICLES} \rightarrow \mathbb{N}$: la position des véhicules.

En fait, cet invariant n'est pas assez puissant pour assurer l'absence de collision. L'analyse des animations et des buts non prouvables a fait ressortir qu'il manque la prise en compte de la vitesse des véhicules. Scheuer et al. (2009) présente et justifie une formule légèrement plus complexe qui évite la collision dans le pire cas (c'est-à-dire le freinage à décélération maximale de deux véhicules voisins). Nous l'avons utilisée pour corriger l'invariant de la machine platoon_2 en :

INVARIANT
$\forall v.(\ v \in 2..\mathsf{VEHICLES} \Rightarrow$
$\mathsf{xpos}(v-1) - \mathsf{xpos}(v)) > \mathsf{CRITICAL\_DISTANCE}$
$+ \max(\{0,(\mathsf{speed}(v-1) * \mathsf{speed}(v-1) - \mathsf{speed}(v) * \mathsf{speed}(v))/(2 * \mathsf{MIN\_ACCEL})\})$

Alors, l'animation se comporte comme prévu et, surtout, la preuve du théorème de vivacité peut être conclue. Le système est dès lors sans blocage, donc sans collision.

7 Conclusion

La relation entre les méthodes formelles et la certification est complexe. Notre expérience indique qu'elle ne saurait se limiter à l'utilisation des démonstrateurs automatiques sur des formules engendrées par les obligations de preuve. Une méthode telle que B doit être utilisée pour donner les fondations et la structure du développement, mais elle doit être complétée pour que la notion de raffinement intègre les activités complémentaires de vérification des propriétés non fonctionnelles et de validation.

Cette question peut être abordée de façon pragmatique. Nous disposons déjà d'outils pour vérifier les contraintes temporelles ou pour effectuer des validations. Notre objectif est d'identifier ceux que nous pouvons employer, de les insérer dans une démarche intégrée de développement et de proposer aux développeurs les outils d'assemblage en un environnement cohérent. Ce papier s'est concentré sur l'identification. Les travaux futurs s'orienteront selon trois axes.

prouvé ? et après ?

Le premier poursuit le développement du système de *platooning* jusqu'au niveau d'une implantation qui pourra être exécutée dans des simulateurs virtuels puis des prototypes réels.

Le deuxième concerne le processus d'animation. Il faut implanter les règles de transformation des spécifications. L'élaboration et l'expression des scénarios reste une question ouverte.

Le troisième axe concerne le problème des contraintes temporelles. A coté de l'emploi des techniques proposées par exemple par Rehm et Cansell (2007) et Rehm (2009), nous menons des investigations avec CSP||B, un formalisme qui combine deux méthodes formelles (Treharne et Schneider (1999)). En usant des techniques présentées par Treharne (2000), nous développons un petit utilitaire pour assister les spécifieurs dans la preuve de cohérence de leur spécifications composites (Nguyen et Jacquot (2010)).

Références

Abrial, J.-R. (1996). *The B Book*. Cambridge University Press.

Abrial, J.-R. (2009). *Modeling in Event-B : System and Software Engineering*. Cambridge University Press.

Balzer, R. M., N. M. Goldman, et D. S. Wile (1982). Operational specification as the basis for rapid prototyping. *SIGSOFT Softw. Eng. Notes 7*(5), 3–16.

Bendisposto, J., M. Leuschel, O. Ligot, et M. Samia (2008). La validation de modèles Event-B avec le plug-in ProB pour Rodin. *Technique et Science Informatiques 27*(8), 1065–1084.

Cansell, D., D. Mery, et J. Rehm (2007). Time Constraint Patterns for Event B Development. In J. Julliand et O. Kouchnarenko (Eds.), *7th International Conference of B Users*, Volume 4355 of *LNCS*, pp. 140–154. Springer Verlag.

Daviet, P. et M. Parent (1996). Longitudinal and lateral servoing of vehicles in a platoon. In *Proceeding of the IEEE Intelligent Vehicles Symposium*, pp. 41–46.

Dijkstra, E. W. (1976). *A Discipline of Programming*. Prentice Hall.

Ferber, J. (1999). *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence*. Addison-wesley Professional.

Ferber, J. et J. P. Muller (1996). Influences and reaction : a model of situated multiagent systems. In *2nd Int. Conf. on Multi-agent Systems*, pp. 72–79.

Lanoix, A. (2008). Event-B specification of a situated multi-agent system : Study of a platoon of vehicles. In *2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 297–304. IEEE Computer Society.

Leuschel, M., L. Adhianto, M. Butler, C. Ferreira, et L. Mikhailov (2001). Animation and model checking of CSP and B using prolog technology. In *Proceedings of the ACM Sigplan Workshop on Verification and Computational Logic VCL'2001*, pp. 97–109.

Mashkoor, A., J.-P. Jacquot, et J. Souquières (2009). Transformation heuristics for formal requirements validation by animation. In *2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems - SafeCert 2009*, Royaume-Uni York.

Nguyen, H. N. et J.-P. Jacquot (2010). A Tool for Checking CSP||B Specifications. In *Proc. Workshop on Tool Building in Formal Methods*, colocated with ABZ Conference – Orford – Canada.

Rehm, J. (2009). *Gestion du temps par le raffinement*. Ph. D. thesis, Nancy-Université — Université Henri Poincaré.

Rehm, J. et D. Cansell (2007). Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method. In *ISoLA*, pp. 179–190.

Rushby, J. (1993). Formal Methods and the Certificaiton of Critical Systems. Techical Report CLS-93-7, Computer Science Laboratory – SRI International.

Scheuer, A., O. Simonin, et F. Charpillet (2009). Safe longitudinal platoons of vehicles without communication. In *ICRA'09 : Proceedings of the 2009 IEEE international conference on Robotics and Automation*, Piscataway, NJ, USA, pp. 2835–2840. IEEE Press.

Schmid, R., J. Ryser, S. Berner, M. Glinz, R. Reutemann, et E. Fahr (2000). A survey of simulation tools for requirements engineering. Technical report, University of Zurich.

Siddiqi, J. I., I. C. Morrey, C. R. Roast, et M. B. Ozcan (1997). Towards quality requirements via animated formal specifications. *Ann. Softw. Eng. 3*, 131–155.

Simonin, O., A. Lanoix, S. Colin, A. Scheuer, et F. Charpillet (2007). Generic Expression in B of the Influence/Reaction Model : Specifying and Verifying Situated Multi-Agent Systems. INRIA Research Report 6304, INRIA.

Treharne, H. (2000). *Combining Control Executives and Software Specifications*. Ph. D. thesis, University of London.

Treharne, H. et S. Schneider (1999). Using a process algebra to control B OPERATIONS. In *1st International Conference on Integrated Formal Methods (IFM'99)*, York, pp. 437–457. Springer Verlag.

Van, H. T., A. van Lamsweerde, P. Massonet, et C. Ponsard (2004). Goal-oriented requirements animation. In *RE '04 : Proceedings of the Requirements Engineering Conference, 12th IEEE International*, Washington, DC, USA, pp. 218–228. IEEE Computer Society.

Summary

In a context of certification, the relationship between formal methods and "correct" software is fuzzy. To formalists, it is the logical consistency (proof), to certification authorities, it is the appropriate fulfilment of needs and constraints of the anticipated usage (validation). We present and discuss an analysis of difficulties and solutions we encountered while writing specifications in Event-B. We propose a development process which takes into account certification issues. It is still based on the refinement of functional properties as supported by Event-B. Refinement steps should be extended with sub-processes whose aims are: to refine the physical and mathematical model to cast it in a form compatible with proof tools, to check non functional and temporal constraints, and to validate the behaviour of the specification.